Design Decisions:

Subclass Structure

* Decided on implementing an abstract Entity class that contains all Plants and Zombies as they share many attributes(Health, Attack, Attack Speed, etc)
* While only one zombie exists now, since more will be added in the future, we created an Abstract zombie class from which our BasicZombie is a subclass.

Level Grid Structure

* To maintain scalability and organization, each level is comprised of lanes. These lanes act almost as independent games from each other since plants and zombies in one lane have no impact on the plants and zombies in another.
* Lanes are further split into spots. Each spot is able to contain up to a single plant (Those close to the right side are typically prevented from hosting plants)
* While Spots are responsible for storing the plants on the game board, since zombies can move, storing them in spots wouldn’t make sense. As a result, zombies are stored within an ArrayList in the lane objects. Initially, a queue would make more sense to store zombies, given that we could easily find out which zombies are in front. In future iterations, zombies will have different movement speeds and be able to pass one another, making this structure meaningless.
* The turn functionality was broken down into each of our classes (The controller calls turn on Level, level calls turn on each lane and lane calls turn on each plant and zombie).

MVC Model

* Changed all code to follow the MVC model to allow for easy development process.
* Original classes from previous milestone leading up to main have been added to model package. They make up the contents of the application.
* Main class from previous milestone split into three classes, Controller, GameCanvas, and Main. Controller handles user input, calls all methods needed to play the game, and updates the View. GameCanvas is the basic look of the level. Main starts the game.
* View class has been added to the view package. It displays the graphical representation of the model, showing where the plants have been placed and where the zombies are.
* Controller communicates the user input to the model, where all the backend damage, movement, sun dollars, and so on are calculated. This information is sent to the view, and the view updates accordingly. Then the code returns back to the controller so the user can make their next input.

Design Decisions Authors: Simon Krol and Shaun Gordon